MODULE *CompleteISSU*

EXTENDS *Naturals*, *FiniteSets*, *Sequences*, *TLC*

The set of all nodes
CONSTANT *Nodes*

The set of all devices
CONSTANT *Devices*

A sequence of all versions
CONSTANT *Versions*

The size of a partition and total number of partitions
CONSTANTS *PartitionSize*, *NumPartitions*

Upgrade states
CONSTANTS *Inactive*, *Initialized*, *Upgraded*, *Committed*, *RolledBack*, *Reset*

An empty value
CONSTANT *Nil*

The current state of the upgrade
VARIABLE *upgradeState*

The current upgrade version
VARIABLE *upgradeVersion*

The set of node states
VARIABLE *nodes*

The set of device states
VARIABLE *devices*

The number of versions must be greater than 1
ASSUME $Len(Versions) > 1$

The number of nodes must be greater than 1
ASSUME $Cardinality(Nodes) > 1$

The partition size is a number
ASSUME $PartitionSize \in Nat \land PartitionSize \leq Cardinality(Nodes)$

The current version domain
$CurrentVersion(v) \triangleq$ CHOOSE $i \in$ DOMAIN $Versions : Versions[i] = v$

Predicate indicating whether the version can be incremented
$CanUpgrade(v) \triangleq CurrentVersion(v) < Len(Versions)$

The next version

```
$NextVersion(v) \triangleq Versions[CurrentVersion(v) + 1]$
```

The previous version

```
$PreviousVersion(v) \triangleq Versions[CurrentVersion(v) - 1]$
```

Return the minimum value from a set

```
$Min(s) \triangleq \text{CHOOSE } x \in s : \forall y \in s : x \leq y$
```

Whether the node has been upgraded

```
$IsUpgraded(n) \triangleq$
    $\land\ nodes[n].version - 1 \in \text{DOMAIN } Versions$
    $\land\ Cardinality(\{i \in Nodes : nodes[i].version = PreviousVersion(nodes[n].version)\}) > 0$
```

Whether other nodes have been upgraded

```
$NotUpgraded(n) \triangleq$
    $\land\ nodes[n].version + 1 \in \text{DOMAIN } Versions$
    $\land\ Cardinality(\{i \in Nodes : nodes[i].version = NextVersion(nodes[n].version)\}) > 0$
```

The type invariant asserts that :
* The total number of versions in the cluster at any given time is ≤ 2
* All devices have a master in the current version

```
$TypeInvariant \triangleq$
    $\land\ Cardinality(\{nodes[n].version : n \in Nodes\}) \in \{1, 2\}$
    $\land\ \forall d \in Devices :$
        LET $master \triangleq devices[d].master$
        IN  $master = Nil \lor nodes[master].version = upgradeVersion$
    $\land\ \forall n \in Nodes :$
        LET $active \triangleq nodes[n].active$
        IN  $\forall p \in \text{DOMAIN } active :$
              IF $IsUpgraded(n)$ THEN
                  $Cardinality(active[p]) = Min(\{Cardinality(\{i \in Nodes : nodes[i].version = nodes[n].version$
              ELSE
                  $Cardinality(active[p]) = PartitionSize$
```

Selects a master node for a device with the given version

```
$ChooseMaster(v, ns) \triangleq$
    LET $choices \triangleq \{n \in ns : nodes[n].version = v\}$
    IN  IF $Cardinality(choices) = 0$ THEN $Nil$ ELSE CHOOSE $n \in choices$ : TRUE
```

Run the command to initialize an upgrade

```
$RunInitialize \triangleq$
    $\land\ upgradeState = Inactive$
    $\land\ \forall n \in Nodes : nodes[n].version = upgradeVersion$
    $\land\ upgradeState' = Initialized$
    $\land$ UNCHANGED $\langle upgradeVersion,\ nodes,\ devices \rangle$
```

Run the command to perform an upgrade

```
$RunUpgrade \triangleq$
    $\land\ upgradeState = Initialized$
    $\land\ CanUpgrade(upgradeVersion)$
    $\land\ Cardinality(\{n \in Nodes : nodes[n].version = NextVersion(upgradeVersion)\}) > 0$
    $\land\ upgradeState' = Upgraded$
    $\land\ upgradeVersion' = NextVersion(upgradeVersion)$
    $\land\ devices' = [d \in Devices \mapsto [master \mapsto ChooseMaster(NextVersion(upgradeVersion),\ Nodes)]]$
    $\land$ UNCHANGED $\langle nodes \rangle$
```

Run the command to rollback an upgrade

```
$RunRollback \triangleq$
    $\land\ upgradeState = Upgraded$
    $\land\ Cardinality(\{n \in Nodes : nodes[n].version = PreviousVersion(upgradeVersion)\}) > 0$
    $\land\ upgradeState' = RolledBack$
    $\land\ upgradeVersion' = PreviousVersion(upgradeVersion)$
    $\land\ devices' = [d \in Devices \mapsto [master \mapsto ChooseMaster(PreviousVersion(upgradeVersion),\ Nodes)]]$
    $\land$ UNCHANGED $\langle nodes \rangle$
```

Run the command to commit a version change

```
$RunCommit \triangleq$
    $\land\ upgradeState = Upgraded$
    $\land\ \forall\, n \in Nodes : nodes[n].version = upgradeVersion$
    $\land\ upgradeState' = Inactive$
    $\land$ UNCHANGED $\langle upgradeVersion,\ nodes,\ devices \rangle$
```

Run the command to reset *ISSU*

```
$RunReset \triangleq$
    $\land\ upgradeState = RolledBack$
    $\land\ \forall\, n \in Nodes : nodes[n].version = upgradeVersion$
    $\land\ upgradeState' = Inactive$
    $\land$ UNCHANGED $\langle upgradeVersion,\ nodes,\ devices \rangle$
```

Helper predicate indicating whether an upgrade can be rolled back during a node state change

```
$CanRollback \triangleq$
    $\land\ upgradeState = Upgraded$
    $\land\ Cardinality(\{n \in Nodes : nodes[n].version = PreviousVersion(upgradeVersion)\}) > 0$
```

Helper to roll back an upgrade during a node state change

```
$RollbackUpgrade \triangleq$
    $\land\ upgradeState' = RolledBack$
    $\land\ upgradeVersion' = PreviousVersion(upgradeVersion)$
    $\land\ devices' = [d \in Devices \mapsto [master \mapsto ChooseMaster(PreviousVersion(upgradeVersion),\ Nodes)]]$
```

Helper to rebalance masters after a node state change

```
$RebalanceMasters(except)\ \triangleq$
    $\land\ devices' = [d \in Devices \mapsto [master \mapsto ChooseMaster(upgradeVersion,\ Nodes \setminus except)]]$
    $\land$ UNCHANGED $\langle upgradeState,\ upgradeVersion \rangle$
```

Helper to determine whether no nodes are in the given version

```
$IsFirstNode(version)\ \triangleq$
    $\land\ Cardinality(\{n \in Nodes : nodes[n].version = version\}) = 0$
```

Upgrade a node in a single atomic step

```
$UpgradeNode(n)\ \triangleq$
    $\land\ \lor\ \land\ upgradeState = Initialized$
            $\land\ nodes[n].version = upgradeVersion$
        $\lor\ \land\ upgradeState = Upgraded$
            $\land\ nodes[n].version = PreviousVersion(upgradeVersion)$
    $\land\ CanUpgrade(nodes[n].version)$
    $\land\ RebalanceMasters($IF $upgradeState = Upgraded$ THEN $\{\}$ ELSE $\{n\})$
    $\land$ IF $IsFirstNode(NextVersion(nodes[n].version))$ THEN
          $nodes' = [nodes$ EXCEPT $![n].version = NextVersion(nodes[n].version),$
                                   $![n].inactive = nodes[n].active,$
                                   $![n].active = [i \in 1\ ..\ NumPartitions \mapsto \{n\}]]$
        ELSE
          LET $active\ \triangleq\ nodes[$CHOOSE $x \in Nodes : nodes[x].version = NextVersion(nodes[n].version)].active$
          IN  $nodes' = [nodes$ EXCEPT $![n].version = NextVersion(nodes[n].version),$
                                       $![n].inactive = nodes[n].active,$
                                       $![n].active = [i \in 1\ ..\ NumPartitions \mapsto$ IF $Cardinality(active[i]) < P$
    $\land$ UNCHANGED $\langle upgradeState,\ upgradeVersion \rangle$
```

Downgrade a node in a single atomic step

```
$DowngradeNode(n)\ \triangleq$
    $\land\ upgradeState = RolledBack$
    $\land\ nodes[n].version = NextVersion(upgradeVersion)$
    $\land\ RebalanceMasters(\{\})$
    $\land\ nodes' = [nodes$ EXCEPT $![n].version = PreviousVersion(nodes[n].version),$
                                 $![n].inactive = [i \in 1\ ..\ NumPartitions \mapsto \{\}],$
                                 $![n].active = nodes[n].inactive]$
    $\land$ UNCHANGED $\langle upgradeState,\ upgradeVersion \rangle$
```

Crash a node in a single atomic step

```
$CrashNode(n)\ \triangleq$
    $\land\ \lor\ \land\ CanRollback$
            $\land\ RollbackUpgrade$
        $\lor\ \land\ \neg CanRollback$
    $\land RebalanceMasters(\{n\})$
    $\land$ UNCHANGED $\langle nodes \rangle$
```

Initial state predicate

```
$Init \triangleq$
    $\land upgradeState = Inactive$
    $\land upgradeVersion = Head(Versions)$
    $\land nodes = [n \in Nodes \mapsto [$
          $version \mapsto Head(Versions),$
          $inactive \mapsto [i \in 1 \,..\, NumPartitions \mapsto \{\}],$
          $active \mapsto [i \in 1 \,..\, NumPartitions \mapsto \text{CHOOSE } x \in \text{SUBSET } (Nodes) : Cardinality(x) = PartitionSize]]$
    $\land devices = [d \in Devices \mapsto [master \mapsto ChooseMaster(upgradeVersion, Nodes)]]$
```

Next state predicate

```
$Next \triangleq$
    $\lor RunInitialize$
    $\lor RunUpgrade$
    $\lor RunCommit$
    $\lor RunRollback$
    $\lor RunReset$
    $\lor \exists\, n \in Nodes :$
        $\lor UpgradeNode(n)$
        $\lor DowngradeNode(n)$
        $\lor CrashNode(n)$
```

\\ ∗ Modification History
\\ ∗ Last modified *Fri Jan* 26 12:57:30 *PST* 2018 by *jordanhalterman*
\\ ∗ Created *Mon Jan* 22 23:16:53 *PST* 2018 by *jordanhalterman*